A45

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 994 425 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
19.04.2000 Bulletin 2000/16

(51) Int. Cl.7: G06F 17/30

(21) Application number: 99112710.1

(22) Date of filing: 30.06.1999

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 30.06.1998 US 91291

(71) Applicant: IOra Ltd.
Woolton Hill, Newbury, Berkshire (GB)

(72) Inventors:
• Draper, Stephen Peter Willis
Basingstoke, Hampshire (GB)
• Collins, Brian James
New Malden, Surrey (GB)
• Falls, Patric Terence
Woolton Hill, Newbury Berkshire (GB)

(74) Representative:
Beyer, Andreas, Dr.
Wuesthoff & Wuesthoff,
Patent- und Rechtsanwälte,
Schweigerstrasse 2
81541 München (DE)

(54) System and method for generating file updates for files stored on read-only media

(57) This invention relates to file updating methods, and more particularly to file updating methods for files stored on Read-Only media. To represent modifications to a set of data objects the method according to the present invention comprises generating a baseline of a first version of a set of data objects; identifying the differences between a second version of the set of data objects and the first version of the set of the data objects; generating update information corresponding to the identified differences, said update information including references to segmented portions of said baseline that correspond to said identified differences regardless of location of said segmented portions in said baseline; and storing the update information whereby the update information may be retrieved and used to generate the second version of the said of data objects from the first version of the said of the data objects.

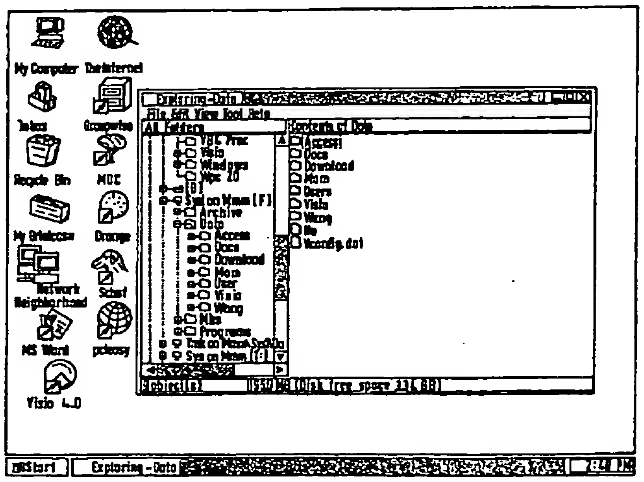


FIG1

Description

Field of the Invention

[0001] This invention relates to file updating methods, and more particularly, to file updating methods for files stored on read-only media.

Background of the Invention

[0002] Software programs and data are frequently distributed on large capacity storage media such as compact disc-read only memories (CD-ROM). These devices are preferably read-only devices to preserve the integrity of the data and program files stored on the device. Such storage devices contain multiple data and executable program files for an application program and typically include a program to install the program and data files on a user's computer. A common application for distributing a computer program and data files is to provide an interface program and data files for employees that use computers remotely from a central site. For example, a company may equip its sales force with CD-ROMs that contain an interface program that retrieves information from the data files. The retrieved data is used to respond to a customer's questions on product availability or product specifications.

[0003] Having timely information available to an organization's remote personnel or being able to provide program patches and other improvements to update software distributed to end users is important for customer service and product support. When the data or programs stored on read-only media are frequently updated or modified, the cost of providing the program or data updates on new storage media such as another CD-ROM can be prohibitively expensive.

[0004] Accordingly, what is needed is a way to update information stored on read-only media without having to produce new read-only media containing the program or data updates for distribution to a company's customers or remote personnel.

Summary of the Invention

[0005] The above limitations of previously known program and data storage read-only storage devices are overcome by a system and method made in accordance with the principles of the present invention. The method of the present invention includes the steps of generating a basis index table identifying the data content of an original file system, generating a file of modification data blocks that may be used to modify the data content of the original file system, and generating a delta look up table for identifying the data blocks in the original file system and the data blocks in the file of modification data blocks that provide the data content for a new version of the original file system. The delta look up table and the file of modification data blocks may be stored for delivery to a computer on which a copy of the original file system is stored. The delta look up table and the file of modification data blocks are then used by the computer system on which a copy of the original file system is stored to provide the data content for a new version of the original file system in way that appears to provide a single file system containing the new version of the file system. Thus, the method of the present invention may be used to generate data for updating the content of a copy of the original file system without having to generate a copy of every file and data block for the new content of the original file system.

[0006] Preferably, the method generates the basis index table by building a basis directory entry meta-data table and a basis index data block table. The basis directory entry meta-data table organizes the meta-data for each entry in a directory enumeration of the original file system by entry name. Preferably, the entry name identifies the entry and its parent.

[0007] The meta-data stored for each entry is known meta-data such as file attributes. The basis index data block table uniquely identifies each data block found within the original file system. For each unique data block identifier, a source file identifier that identifies the source file for the data block, the offset to the first data unit for the block within the source file, and the length of the data block are stored. These two tables may then be used to generate the files for generating a new version of the file system.

[0008] The method of the present invention also includes the steps of generating a delta directory map file to identify the structure of the entries in the new version of the original file system, a delta look up table (LUT) file for identifying the location of the data blocks to generate the files in the new version of the original file system, and a delta modification data block file that contains the new data content for the new version of the original file system. The delta directory map file contains the name for the entries in the new version of the original file system, the modification status for the entries in the new version of the file system, the meta-data for each entry having a modification status of "modified", "contents modified" or "new", the first look up table record for each file entry, and the number of look up table records used to construct the file in the new version of the original file system. The delta look up table contains at least one LUT record for each file entry having a modification status of "contents modified" or "new". A LUT record identifies the source file containing the data block, the location of the first data unit of the data block in the identified source file, the length of the data block, and the offset of the first data unit of the data block in the file being processed. The source file identifier either identifies a file of the original file system or the modification data block file for the new version. The LUT records for all of the files in the new version of the original file system are stored in an LUT file. The location of the first LUT record for a file is identified by a pointer stored in the meta-data of the delta directory map file for file entries having a modification status of "contents modified" or "new". The directory map file may also be used in the computer having a copy of the original file system to generate information for display or use regarding the structure of the new version of the original file system and its new data content. Structure data includes data that is displayed in response to a directory enumeration command or the like. The delta modification data block file contains the data blocks having new data content for the file entries of the new version of the original file system. As the data blocks for the new data content of the new version of the original file system are stored in the delta modification data block file, a delta index data block table is generated. This table includes a unique identifier for each data block stored in the delta data block modification file that has unique data content, an identifier that indicates the version of the delta data block modification file that is the source file for the block, the offset to the first data unit for the data block and the length of the data block. The delta index data block table is appended to the basis index data block table.

[0009] The delta directory map file, the delta modification data block file and the delta look up table may be compressed and stored on storage media or downloaded to a computer having a copy of the original file system. The downloaded delta directory map file, the delta modification data block file and the delta look up table file are used to seamlessly regenerate a new version of the original file system. This regeneration of the new version of the original file system is done in a manner which gives the appearance that the contents of the device on which a copy of the original file system is stored have been modified, even if the device uses read-only media for storage of the original file system. Thus, the system and method of the present invention provide a mechanism for updating the contents of a file system without requiring the production of a complete file system corresponding to the new version of the program and/or data stored in the file system.

[0010] Preferably, the method and system of the present invention may be used to generate a representation of a new version of an original file system with reference to the original file system and to the delta modification data block files for previous versions of the original file system. This use of previous versions reduces the amount of data to be stored in the delta directory map file, delta modification data block file, and delta look up table for the latest version of an original file system. In this embodiment of the present invention, the process for generating the files for the new version of the original file system produces delta data block records that identify the source file for a data block as being either a file in the original file system, a delta modification data block tile for a previous version of the original file system or the delta modification data block file for the new version of the original file system. The version of the delta modification data block table containing a data block is determined from the delta index data block tables appended to the basis index data block table.

[0011] These and other benefits and advantages of the present invention shall become apparent from the detailed description of the invention presented below in conjunction with the figures accompanying the description.

Brief Description of the Drawings

[0012]

Fig. 1 is a depiction of a screen shot of a file system hierarchy that may be evaluated by the system of the present invention;

Fig. 2 is a flowchart of an exemplary process that generates a representation of the original file system;

Figs. 3A and 3B are a flowchart of an exemplary process that generates the look-up table file and modification data block file for an update to the representation of the original file system generated by the process shown in Fig. 2;

Fig. 4 is a flowchart of an exemplary process that generates a delta directory map file for the new version of the original file system from the delta directory entry meta-data table generated by the process shown in Figs. 3A and 3B; and

Fig. 5 is a flowchart of an exemplary process that uses the files for an update generated by the process shown in Figs. 3A, 3B and 4 to generate a latest version of the original file system.

Detailed Description of the Invention

[0013] Fig. 1 depicts a screen shot of a file system hierarchy. The hierarchy for the file system is comprised of a directory having a list of file entries and subdirectory entries. The subdirectory entries may include additional files for the file system. Each entry in the directory for the file system hierarchy also contains meta-data. For the file entries the meta-data includes known file meta-data such as the file name, file attributes, and other known file meta-data.

[0014] In order to generate modification data files for a file system hierarchy, the original version of the file system hierarchy is processed and information about the system is stored in a file system map file. This process is depicted in Fig. 2. The process begins by processing the directory file for the highest level of the file system hierarchy to identify the entries for the subdirectories and files at the highest level in the file system

(Block 50). For each entry, meta-data for the entry is stored in a basis directory meta-data table (Block 54). If the entry is a subdirectory (Block 56), the process determines whether another directory entry exists for processing (Block 90). If there is, it is processed (Block 54). Otherwise, the process terminates as the basis directory entry meta-data table and basis index data block table have been generated.

[0015] For a file entry being processed, the file is segmented into blocks of one or more fixed lengths (Block 60). Preferably, the block length or lengths are chosen so that the whole basis index data block table can be held in the memory of the computer. In this way every block of memory can be directly and efficiently accessed. For this reason the block length should be determined as a function of the available computer resources.

[0016] For each block, an iterative checksum (Block 62) and then a safe checksum (Block 64) is generated. The iterative checksum is a value that is computed from the data values for each byte within a block beginning at the first byte of the block to the last byte in the block. It possesses the property that an iterative checksum for a data block comprised of the first N data units in a data string may be used to generate the iterative checksum for the next data block comprised of the N data units beginning at the second data byte. This is done by performing the inverse iterative checksum operation on the iterative checksum using the data content of the first data unit of the first block to remove its contribution to the iterative checksum and then performing the iterative checksum operation on the resulting value using the N+1 data unit that forms the last data unit for the next data block. Thus, two data operations may be used to generate the iterative checksum for the next block in a data string in which the successive data blocks are formed by using a sliding data window in the data string. For example, an addition operation may be used to generate an iterative checksum having the property noted above. A safe checksum is generated by a process that is less likely to produce the same checksum for two blocks having different data contents than the storage media is likely to return an inaccurate data value. A safe checksum generation method well known within the data communication art is the MD5 checksum. The iterative and safe checksum pair for a data block form a checksum identifier that is used to identify the data block. The iterative checksum is not as computationally complex as the safe checksum so the iterative checksum is a relatively computational resource efficient method for a determining that two data blocks may be the same. The safe checksum may then be used to verify that the data content of the blocks are the same and reduce the likelihood of a false positive identification. If the checksum identifier is the same as the checksum identifier for a data block previously stored in the index data block table (Block 68) then the data content of the data block is not unique. Thus, the data block record in the index data block table for the corresponding checksum identifier adequately defines the data block being processed so the checksum identifier is not stored in the index data block table and the process determines whether another data block is to be processed (Block 82).

[0017] If the checksum identifier indicates the data block content is unique, the iterative checksum is stored as the primary key in the index data block table and the safe checksum is stored in the index data block table as a qualified key (Block 70). Associated with the checksum identifier for the block is an identifier for the file from which the data block came (Block 74), the offset from the first byte within the file to the first byte in the data block (Block 76), and the length of the data block (Block 78). The source file identifier may be the name of the file in which the data block is stored, but preferably, it is a pointer to the meta-data in the basis directory entry meta-data table for the source file. This process of identifying and storing information about each data block in the index data block table continues (Block 82) until all of the blocks for a file entry have been processed. A safe checksum for the entire data content of the file is then generated and stored in the basis directory entry meta-data table (Block 84). The process continues (Block 90) until all entries for the entire directory structure for the original file system have been processed. The basis directory entry meta-data table and basis index data block table, file system map file representing the meta-data and data content for each entry within the file system hierarchy is then stored on storage media (Block 96). This data forms the baseline for generating modification data files for updating the original file system.

[0018] Whenever a new version of a file system hierarchy is generated, either by changing, deleting or adding data to a file or its meta-data or by adding or deleting data files to the file system, a delta modification data block file and delta look up table may be generated to provide the update information for the differences between the original file system hierarchy and the new version of the file system hierarchy. The process for generating the delta modification data block file and the delta look up table is shown in Fig. 3. That process begins by reading the directory file for the new file system hierarchy and identifying the entries for the subdirectories and files in the file system hierarchy (Block 100). Each entry is then processed by storing the meta-data for the entry in a delta directory entry meta-data table (Block 104). The status of the entry is then determined by searching the basis directory entry meta-data table for an entry having the same name under the same parent (Block 108). If no corresponding entry is located in the basis directory entry meta-data table (Block 110), then the modification status for the entry in the new file system hierarchy is set to "new" (Block 112). If a corresponding entry is located in the basis directory entry meta-data table then the meta-data for the corresponding entry is compared to the meta-data for the entry in the delta directory entry meta-data table (Block 114) and, if the meta-data is the same for both entries, the modification status is set to "unmodified" (Block 116). If the meta-data for the entries do not correspond and the entries are not files (Block 120), the modification status is set to "modified" (Block 122). If the meta-data for the entries do not correspond and the entries are files, a safe checksum is generated for the data contents of the file entry in the new file system (Block 126). This safe checksum is compared to the safe checksum for the entire data content of the file stored in the basis directory entry meta-data table (Block 128) and if they are not equal, the modification status is set to "contents modified" (Block 130). Otherwise, the modification status is set to "modified" (Block 134). The modification status is stored in the delta directory entry meta-data table. This process continues until all of the entries in the new version of the original file system have been processed (Block 136).

[0019] The basis directory entry meta-data table is now searched to determine whether a corresponding entry exists in the delta directory entry meta-data table. Specifically, a directory entry in the basis directory entry meta-data table is selected (Block 140) and the delta directory entry meta-data table is searched for a corresponding entry (Block 142). If no corresponding entry is located, an identifier for the entry and a modification status of "deleted" is generated and stored in the delta directory entry meta-data table (Block 144). The process continues until all entries in the basis index directory entry meta-data table have been checked (Block 146).

[0020] The process now selects an entry in the delta directory entry meta-data table (Block 150) and determines whether it has a modification status of "new" or "contents modified" (Block 152). For these entries, look up table (LUT) records are generated and data blocks stored in the delta modification data block file, if necessary. If an entry is identified as being a "new" or "contents modified" entry, a sliding window of N data units, such as 256 bytes, is used to define data blocks (Block 156). As noted before, the number N must be one of the block sizes used to segment files in the original file system for constructing the basis index data block table. An iterative checksum is computed for the first data block formed by the sliding window being placed at the first data unit of the data contents of the "new" or "contents modified" file (Block 158). This iterative checksum is compared to the iterative checksums of the checksum identifiers stored in the basis index data block table to determine whether a corresponding entry may exist (Block 160). If no corresponding iterative checksum is found, the checksum identifier for the data block being processed cannot be the same as one in the index basis data block table so the first data unit of the data block in the sliding window is stored in a delta modification data block file (Block 162). The sliding window is then moved to remove the first data unit from the data block in the file being processed and to add the next data unit (Block 156). The iterative checksum for the data block in the sliding window is computed (Block 158) and compared to the iterative checksums of the checksum identifiers in the basis index data block table(Block 160). Because the iterative checksum has the property discussed above, the iterative checksum for each successive data block only requires calculations to remove the contribution of the data units removed from the block by moving the sliding window and to add the contributions of the data units added by moving the sliding window. Moving the sliding window, generating the next iterative checksum and comparing the generated iterative checksum to those for the checksum identifiers in the basis index data block table continues until a corresponding iterative checksum for one of the checksum identifiers is located or the number of data units stored to the delta modification data block file corresponds to the number of data units for a data block (Block 172). When a data block of modification data has been stored to the delta modification data block file, the iterative and safe checksums for the block are generated to form a checksum identifier for the block (Block 174). The iterative checksum and safe checksum for the block of modification data are then stored as the primary key and qualified key, respectively, in a delta index data block table associated with the new version of the original file system. An identifier of the delta modification data block file in which the data block is stored, the offset into that file that defines the location of the first data unit for the data block being processed, and the length of the data block being processed are also stored in the delta index data block table in association with the iterative and safe checksums (Block 176).

[0021] Once an iterative checksum for a data block within the sliding window corresponds to one or more iterative checksums in the checksum identifiers stored in the basis index data block table, the process computes the safe checksum for the block within the sliding window and compares it to the safe checksums of the checksum identifiers selected from the basis index data block table (Block 178). Only one, if any, safe checksum of the checksum identifiers should be the same as the safe checksum computed for the data block. If a corresponding safe checksum is identified, the data blocks are the same. The process determines whether the previous data block checksum identifier comparison indicated a corresponding checksum identifier in the basis index data block table was located (Block 180). If the previous checksum identifier comparison did not find a corresponding checksum identifier, a look up table (LUT) record is generated for the data units stored in the delta modification data block file since the last corresponding checksum identifier was detected (Block 182). That is, all of the data following the identification of the last data block that is also in the basis index data block table is stored in the delta data modification file and the LUT record for that data indicates that the data is a contiguous block of data. The LUT record is comprised of a delta modification data block file identifier, the offset from the first data unit in the modification data file to the contiguous data block stored in the modification data file, the number of data units in the contiguous data block stored in the modification data file, and the offset of the data block in the file currently being processed. The first three data elements in the LUT to identify the source file for the data block in the new version of the original file system and its location in that file while the fourth data element defines the location of the data block in the file of the new version of the original file system. As discussed below, this permits the application program that controls access to the new version of the original file system to not only know from where it can retrieve the data block but where it goes in the new version of the file.

[0022] At this point in the process, the checksum identifier for the data block within the sliding window has been identified as being the same as a checksum identifier in the basis index data block table. As this block already exists in a file in the original version of the file system, a different LUT record is generated for the data block within the sliding window (Block 198). The LUT record for the data block that corresponds to the checksum identifier stored in the basis index data block table is comprised of the same source file identifier as the one in the basis index data block table, the same offset from the start of the source file, the same data block length stored in the basis index data block table, and the offset of the data block in the file currently being processed. The process then continues by determining whether the previous LUT record for the file being processed has a source file identifier that is the same as the one for the LUT record generated for the data block within the sliding window (Block 200). If it does and the LUT just generated is for a data block that is contiguous with the data block identified by the previous LUT record, the process increases the length stored in the previous LUT record by the length of the data block in the LUT record generated for the data block just processed and discards the new LUT record (Block 202). This corresponds to the situation where contiguous blocks of the data in a file of the new version of the original file system are the same as a group of contiguous blocks in a file of the original file system. Thus, one LUT record can identify a source for the contiguous group of blocks. If the data block for the new LUT record is not contiguous with the data block of the previous LUT record or is not from the same source file, then the LUT record is appended to the previous LUT record (Block 206). If the safe checksum does not correspond to the safe checksum for a data block having the corresponding iterative checksum, the process determines whether a data block of modification data has been defined (Block 172). The process continues until it determines whether all data units in the file have been processed (Block 210). If more data units exist, the sliding window is moved by its length to capture a new data block (Block 212). If the number of remaining data units do not fill the sliding window (Block 214), the remaining data units are stored in the delta modification data block file (Block 218) and a corresponding LUT record is generated (Block 220). The LUT records generated for the file being processed are then appended to the LUT records for other files previously stored in an LUT file for the new version of the original file system (Block 222) and the LUT records for the file are stored in the LUT file (Block 224). The offset for the first LUT for the file being processed and the number of LUT records for this file are then stored in the meta-data of the delta directory entry meta-data table for the file being processed (Block 228). The process then checks for more entries in the delta directory entry meta-data table to process (Block 230). If there are more entries the process continues (Block 150). If all of the delta directory entries have been processed, the delta index data block table is appended to the basis index data block table (Block 234) and the delta directory entry meta-data table for the entries in the new version of the original file system are then searched for any entries having a modification status of "unmodified". These entries and their meta-data are removed unless they have a descendant having a modification status other than "unmodified" (Block 238).

[0023] In an embodiment of the present invention that utilizes previous updates provided for the original file system, the above process is modified to evaluate the delta index data block tables for previous versions of the original file system. Specifically, the process searches the basis index data block tables and the delta index data block tables files for update versions to locate data blocks having corresponding iterative and safe checksums for corresponding "new" or "contents modified" files in the latest version. Additionally, the source of data blocks may also include delta modification data files for previous update versions of the original file system as well as the files of the original file system and the delta modification data block file for the latest version.

[0024] The delta directory entry meta-data table for the new version of the original file system generated by the process in Fig. 3 is then used by the process shown in Fig. 4 to generate a delta directory map file. An entry is selected from the delta directory entry meta-data table (Block 250) and an entry in the delta directory map file system is generated. The entry at least includes the name of the entry (Block 254) and its modification status (Block 256). If the modification status is "new", "modified" or "contents modified" (Block 260), the new meta-data is also stored in the delta directory map file for the entry (Block 264). If the modification status is "new" or "contents modified", (Block 266), the offset to the first LUT record for the file in the LUT file and the number of LUT records for the file in the LUT file are stored in the delta directory map file (Block 268). The process continues until all entries in the delta directory entry meta-data table have been processed (Block 270). The name of the new file system hierarchy, its version identifier, directory map file, LUT file, and modification data files may now be compressed for delivery to a system having a copy of the original file system.

[0025] Once the compressed representation of the new version of the original file system is transferred to a computer on which a copy of the original file system hierarchy is stored, it may be used to update the original file system. An application program may be provided as part of that representation to perform the process depicted in Fig. 5. Alternatively, the application program may be part of the interface program provided for accessing the content of the original file system hierarchy such as an extension to the file system program of the recipient computer. The program decompresses the representation of the new file system hierarchy and stores the delta directory map file, LUT file, and delta modification data block file in storage accessible to the computer. The process then determines whether a directory containing a delta modification data block file for a previous version of the original file system hierarchy is associated with a directory or drive containing the original file system hierarchy (Block 300). If there is an association with a directory containing a delta modification data block file, that association is merged with an association between the directory where the decompressed files for the new file system hierarchy are stored and the drive or directory where the original file system hierarchy is stored (Block 302).

[0026] The merge replaces the existing associated delta directory map file and LUT file with the new delta directory map file and LUT file, but leaves any existing delta modification data block files referenced in the new LUT file. In other words, when a representation of a new version of a file system hierarchy is transferred to a computer to update a copy of the original file system hierarchy, the process determines if there is an existing association with a directory containing a delta modification data block file. If there is such an association, then that association is merged with that of the new version and the merge replaces the existing delta directory map file and LUT file with those of the new version (Block 302).

[0027] Alternatively, the replaced delta directory map file and LUT file from the previous association could instead be retained in addition to the new files. With this alternative, the process could allow the user to select which of a number of available versions of a file system hierarchy is accessed when the user attempts to access the original file system hierarchy. Such a selection mechanism provides an accessible archive of multiple versions of the file system hierarchy.

[0028] Otherwise, an association between the drive or directory where the original file system hierarchy is stored and the directory where the downloaded decompressed files for the new version of the original file system hierarchy is now located is generated (Block 308). The application program may be coupled to the operating system of the computer in which a copy of the original file system hierarchy and the decompressed files for the new version of the file system hierarchy are stored. In a known manner, the operating system is modified to detect any attempted access to the drive or directory containing the original file system hierarchy or the files for the new version of the file system hierarchy. In response to an attempted operation to change the physical media for the original file system hierarchy (Block 310), the application program stores a media change indicator (Block 314) and verifies the identity of the physical media when a subsequent attempt is made to access the original file system hierarchy (Block 318). If the physical media has changed, the application change program checks the media change indicator and determines whether the original file system media is available. If it is not, the program indicates that the original file system hierarchy is not available for access by the user. Otherwise, the access is processed. Attempts to write data to the drive or directory containing the original file system hierarchy or the files for the new version of the original file system detected by the application program (Block 320) are not processed (Block 324).

[0029] For commands attempting to interrogate the structure of the original file system hierarchy, the application program responds by building data in two passes and presenting that data to the user. A command to interrogate the structure of the original file system hierarchy is one such as a directory enumeration command. In response to a structure inquiry (Block 328), the application program first retrieves the requested structure data from the original file system and deletes the entries for which the modification status in the delta directory map file is "deleted", "modified", "new" or "contents modified". The data for these entries is obtained from the delta directory map file and used to modify the structure data responsive to the structure query (Block 330). That is, the application program obtains the data to be displayed for the original file system hierarchy, deletes those files corresponding to delta directory map file entries having a modification status of "deleted", adding structure data for those entries in the directory map file having a status of "new", and modifying the structure data for those entries in the directory map file having a status of "modified" or "contents modified". This data is then provided to the operating system for display to the user.

[0030] For file system operations that open a file in the new version of the original file system hierarchy (Block 340), the application program determines whether the modification status of the file is "unmodified". If it is, the operation is processed using the contents of the original file system only. Otherwise, the application program constructs and returns an open file handle that identifies the file (Block 344). The open file handle identifies the file for subsequent file operation commands but does not open any underlying file. For any file system operation command that interrogates the properties of a file for which an open file handle exists, the application program returns data from the delta directory map file entries that correspond to the file identified by the open file handle.

[0031] In response to an I/O operation command that reads data from a file identified by an open file handle (Block 350), the application program constructs a response to the query by identifying the LUT record in the LUT file that corresponds to the start of the requested data (Block 352). If the underlying file referenced in the LUT record is not opened, the application program opens the underlying file and associates it with the open file handle. The program then reads from the LUT record whether the data for the requested data block is to be read from the original file system hierarchy or one of the delta modification data block files. After the source file is identified, the offset data and data block length are used to locate the first byte to be transferred from the identified source file and the number of bytes to be transferred, respectively. The corresponding number of bytes are transferred from the source file to a response being built (Block 356). If additional data is required for the response (Block 360), the next LUT record is used to extract data for the response (Block 364). This process continues until the data transferred for an LUT record provides all of the data requested or until the last entry for the file is reached. The response built from the transfer of data from the source files identified by the LUT records is then provided to the operating system for delivery to the requesting program (Block 368). In this manner, a response is provided to a file system operation that appears to be the result of a single contiguous read operation. In response to a file system operation that closes a data file (Block 370), the application program closes all corresponding files in the original file system hierarchy and the data files for the new file system hierarchy (Block 372).

[0032] In the above description, a delta index data block table is constructed to contain a delta index data block record for each new block of modification data (Block 176). When all the delta directory entries have been processed, the delta index data block table is appended to the basis index data block table (Block 234).

[0033] Alternatively (at Block 176), the basis index data block table could be updated to contain a basis index data block record for each new block of modification data as that block of modification data is processed. In this way, there would be no delta index data block table and the step at Block 234 would be eliminated.

[0034] With this alternative, the new version of the file system hierarchy can contain new or modified files in which the same new block of modification data appears more than once, but the generated representation of the new version of the original file system hierarchy will only contain a single copy of the new block of modification data. A significant special case of this is where the original file system hierarchy is empty and the method of this invention then generates an efficiently compressed representation of a file system hierarchy.

[0035] The method of the invention is described for versions of a file system hierarchy and the files holding data contents. The method may also be applied to any structure of identified objects which contain data. One further example of such a hierarchy is a Directory Services hierarchy representing objects used to manage a computer network. Other similar examples would be obvious to those skilled in the art.

[0036] The method of the invention generates a compact representation of the differences between an original version of a file system hierarchy and an updated version of the file system hierarchy, and allows the regeneration of the updated version of the file system hierarchy from the original version of a file system hierarchy using that generated representation. There are many other uses of such a method. One such use is to back up a file system hierarchy or updates to the file system hierarchy to allow that version to be restored at a later date. In this case, the sequence of generated representations of the differences between the versions of the file system hierarchy could be used to restore any version. Other similar examples would be obvious to those skilled in the art.

Claims

1. A method for representing modifications to a set of data objects comprising:

   - generating a baseline of a first version of a set of data objects;
   - identifying the differences between a second version of the set of data objects and the first version of the set of data objects;
   - generating update information corresponding to the identified differences, said update information including references to segmented portions of said baseline that correspond to said identified differences regardless of location of said segmented portions in said baseline; and
   - storing the update information whereby the update information may be retrieved and used to generate the second version of the set of data objects from the first version of the set of data objects.

2. The method of claim 1, said baseline being generated by forming a basis index data block table file system map from the first version of the set of data objects.

3. The method of claim 2, said baseline generation further includes forming a basis directory entry meta-data table from the first version of the set of

data objects.

4. The method of claim 1 wherein said first version of said set of data objects is a directory hierarchy of a file system and the data objects are files.

5. The method of claim 1 wherein said first version of said set of data objects is a directory services hierarchy of data objects used to manage a computer network

6. The method of claim 1 said generation of said baseline further comprising:

- selecting a data object of said first version of said set of data objects;
- generating and storing identifying data corresponding to the identity of said selected data object;
- segmenting each selected data object into blocks to form segmented portions;
- determining whether the data content of each block is unique with respect to other data blocks segmented from data objects previously selected;
- storing data identifying the content of the data block and its location within a corresponding data object to form said segmented portions for said baseline;
- continuing the selection, segmentation, determination and storing of data for data blocks within said data objects of said first version of the set of data objects to generate the baseline until all data objects within said first version of said set of data objects have been selected.

7. The method of claim 6 wherein a size of said blocks into which said selected data objects are segmented is a length determined with reference to the available computer resources.

8. The method of claim 6 or 7 wherein said data identifying the content of data blocks and their locations are stored within a basis index data block table file system map.

9. The method of claim 8 wherein said identifying data corresponding to each data object are stored in a meta-data table of said baseline.

10. The method of claim 1 wherein said differences between said first and said second versions are identified by

- identifying new data objects in said second version of said set of data objects by determining whether data objects in said second version of said set of data objects are in said first version;
- identifying modified data objects in said second version of said set of data objects by determining whether said data objects in both said first and second versions are the same;
- identifying deleted data objects by determining the absence of said first version data objects in said second version of said set of data objects; and
- storing data identifying said new data objects, said modified data objects and said deleted data objects in said update information.

11. The method of claim 6 wherein a size of said blocks into which said selected data objects are segmented is a fixed length.

12. The method of claim 10 wherein said generation of said update information includes:

- generating a delta modification data block file; and
- generating a delta look-up table.

13. The method of claim 12 wherein said identification of new data objects, modified data objects and deleted data objects includes:

- comparing meta-data for data objects in the second version of the set of data objects to meta-data for data objects in said baseline to determine whether a data object in said second version of said set of data objects is a new data object or a modified data object; and
- comparing meta-data for data objects in said baseline to meta-data for data objects in said second version of said data objects to determine whether one of said data objects in said first version of said data objects is absent in said second set of said data objects.

14. The method of claim 13 wherein said meta-data for data objects in the first version of the set of data objects are stored in a basis directory entry meta-data table; and

- said meta-data for data objects in the second version of the set of data objects are stored in a delta directory entry meta-data table.

15. The method of claim 14 further comprising:

- generating a delta directory map file from said delta directory entry meta-data table and said delta look-up table.

16. The method of claim 1 further comprising:

- segmenting data objects in said second version;

- identifying whether said segmented portions of said data objects correspond to segmented portions of said data objects in said baseline; and
- storing each said data block and a corresponding identifier for each said segmented portion in said second version for which no corresponding segmented portion was identified in said baseline.

17. The method of claim 16 further comprising:

- identifying a contiguous segmented portion in the second version of said set of data objects that corresponds to a contiguous segmented portion in said baseline.

18. The method of claim 16 wherein said storage of said identifiers includes:

- generating an iterative checksum from a segmented portion of said data object;
- generating a safe checksum from said segmented portion of said data object; and
- forming the identifier from said iterative checksum and said safe checksum.

19. The method of claim 16 wherein said identification of said corresponding segmented portions in said first and said second versions of said sets of data objects include:

- comparing an iterative checksum for said segmented portion in said second version of said set of data objects to an iterative checksum for said segmented portion in said baseline; and
- comparing a safe checksum for said segmented portion in said second version of said set of data objects to a safe checksum for said segmented portion in said baseline in response to said iterative checksum comparison indicating said iterative checksums correspond.

20. The method of claim 1 further comprising:

- identifying differences between a new version of the set of data objects and the baseline and the update information for all intervening versions of the set of data objects;
- generating update information corresponding to the identified differences; and
- storing the update information whereby the update information may be retrieved and used to generate the new version of the set of data objects from the baseline and the update information of the intervening and new sets of data objects.

21. The method of claim 20 wherein said generation of update information may include references to segmented portions of said baseline or to the update information for any intervening version of the set of data objects, said references being stored in said update information for said new version so that segmented portions of said new version of the set of data objects that occurred in the baseline or any intervening version of the set of data objects are not stored in said update information for the new version of the set of data objects.

22. The method of claim 20 or 21 wherein said generation of update information may include references to segmented portions or to the update information for said new version of the set of data objects, said references being stored in said update information for said new version so that segmented portions of said new version of the set of data objects in said new version of the set of data objects are not stored more than once in said update information for the new version of the set of data objects.

23. A method for providing data associated with a data object contained within an updated set of data objects stored on a computer that includes:

- representing a hierarchical set of data objects with a baseline corresponding to an original version of the hierarchical set of data objects and update information corresponding to differences between the original version and a new version of the hierarchical set of data objects;
- responding to a data access operation requesting access to a data object within the hierarchical set of data objects by designating one of the baseline and the update information as a source for at least a portion of the requested data object and identifying the data to be retrieved from the source that corresponds to the data object;
- retrieving the identified data from the designated source; and
- continuing to designate one of the baseline and the update information as the source for additional portions of the requested data object, to identify the data to be retrieved from the designated source and to retrieve the identified data until all data for the requested data object has been retrieved whereby data objects requested by data access operations may be represented by data stored in both the baseline and the update information representing the hierarchical set of data objects.

24. The method of claim 23 where multiple versions of said update information may be stored and any one such version may be selected at any time to determine which updated version of said hierarchical set of data objects is retrieved in response to said data access operation.

25. The method of claim 23 or 24 in which the response to a data access operation further includes:

- determining whether the data access operation accesses a data object represented only by the baseline; and
- generating an identifier for a data object identified by the data access operation in response to a determination that the data object is represented by the baseline and the update information.

26. The method of claim 23 which further includes:

- not processing data access operations that attempt to write data to the baseline or the update information.

27. The method of claim 23 which further includes:

- responding to a structure inquiry by retrieving structure data from the baseline; and
- modifying the structure data retrieved from the baseline with the update information for the new version of the hierarchical set of data objects.

28. The method of claim 27 which further includes:

- identifying a modification status in the update information for data objects identified by the structure inquiry;
- deleting the structure data for data objects having a deletion modification status;
- adding the structure data in the update information for data objects having an added modification status; and
- modifying the structure data with data from the update information for data objects having modified status.

29. The method of claim 23 which further includes:

- storing the update information in a delta directory map file, a look-up table, and at least one delta modification data block file.

30. The method of claim 23 which further includes:

- storing structure data for the set of hierarchical set of data objects in the baseline and the update information; and
- storing data content of the data objects in the baseline and the update information, the data content of the data objects being separate from the structure data.

31. The method of claim 30 wherein the storage of structure data includes storing of directory hierarchical data; and

- the storage of data content of the data objects includes storing of data within the data objects in the hierarchical set of data objects.

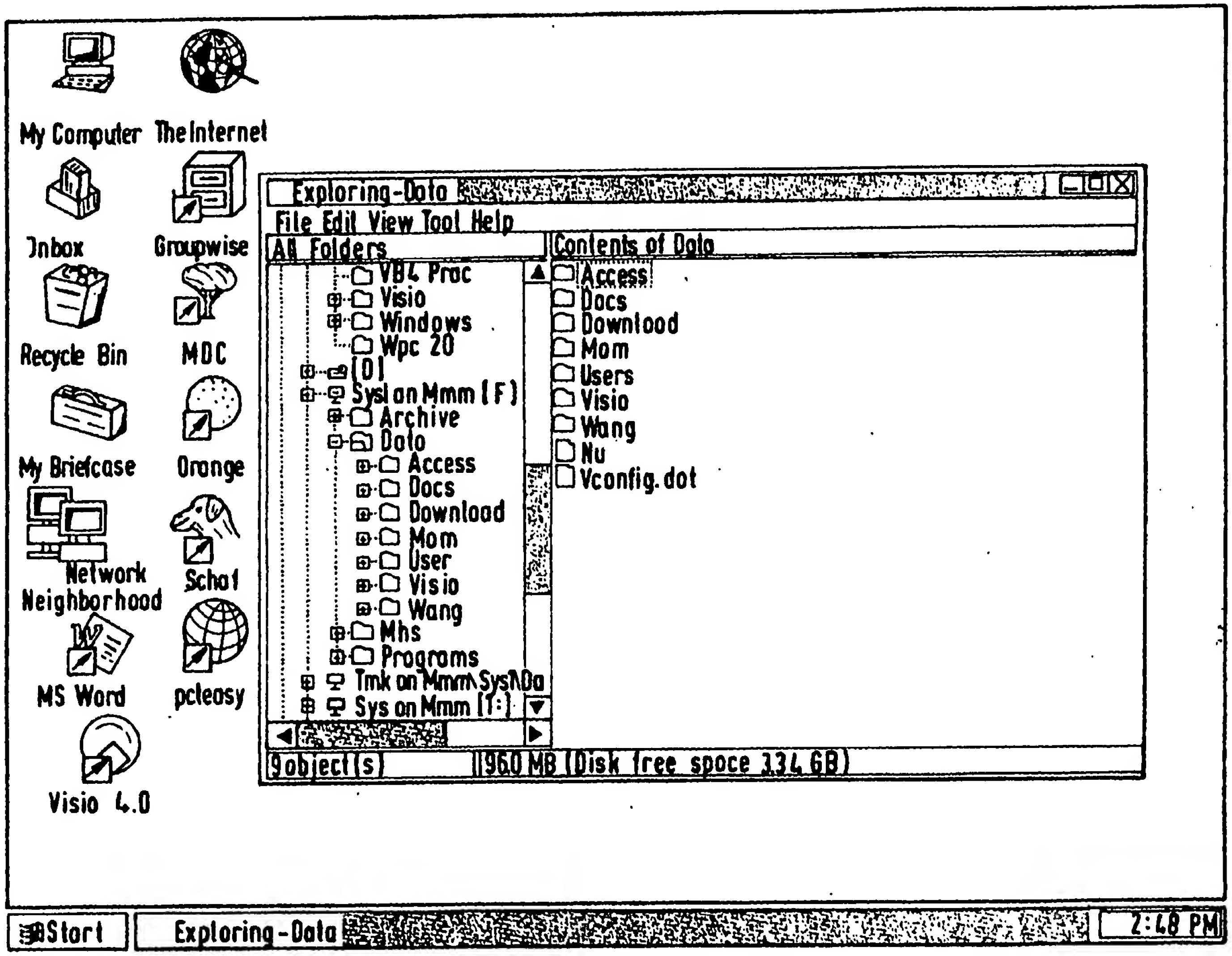

My Computer
The Internet
Inbox
Groupwise
Recycle Bin
MDC
My Briefcase
Orange
Network Neighborhood
Schol
MS Word
pcleasy
Visio 4.0
Exploring-Data
File Edit View Tool Help
All Folders
VB4 Proc
Visio
Windows
Wpc 20
(D)
Sys1 on Mmm (F)
Archive
Data
Access
Docs
Download
Mom
User
Visio
Wang
Mhs
Programs
Tmk on Mmm\SysData
Sys on Mmm (I:)
Contents of Data
Access
Docs
Download
Mom
Users
Visio
Wang
Nu
Vconfig.dot
9 object(s)
1960 MB (Disk free space 334 GB)
Start
Exploring-Data
2:48 PM

FIG.1

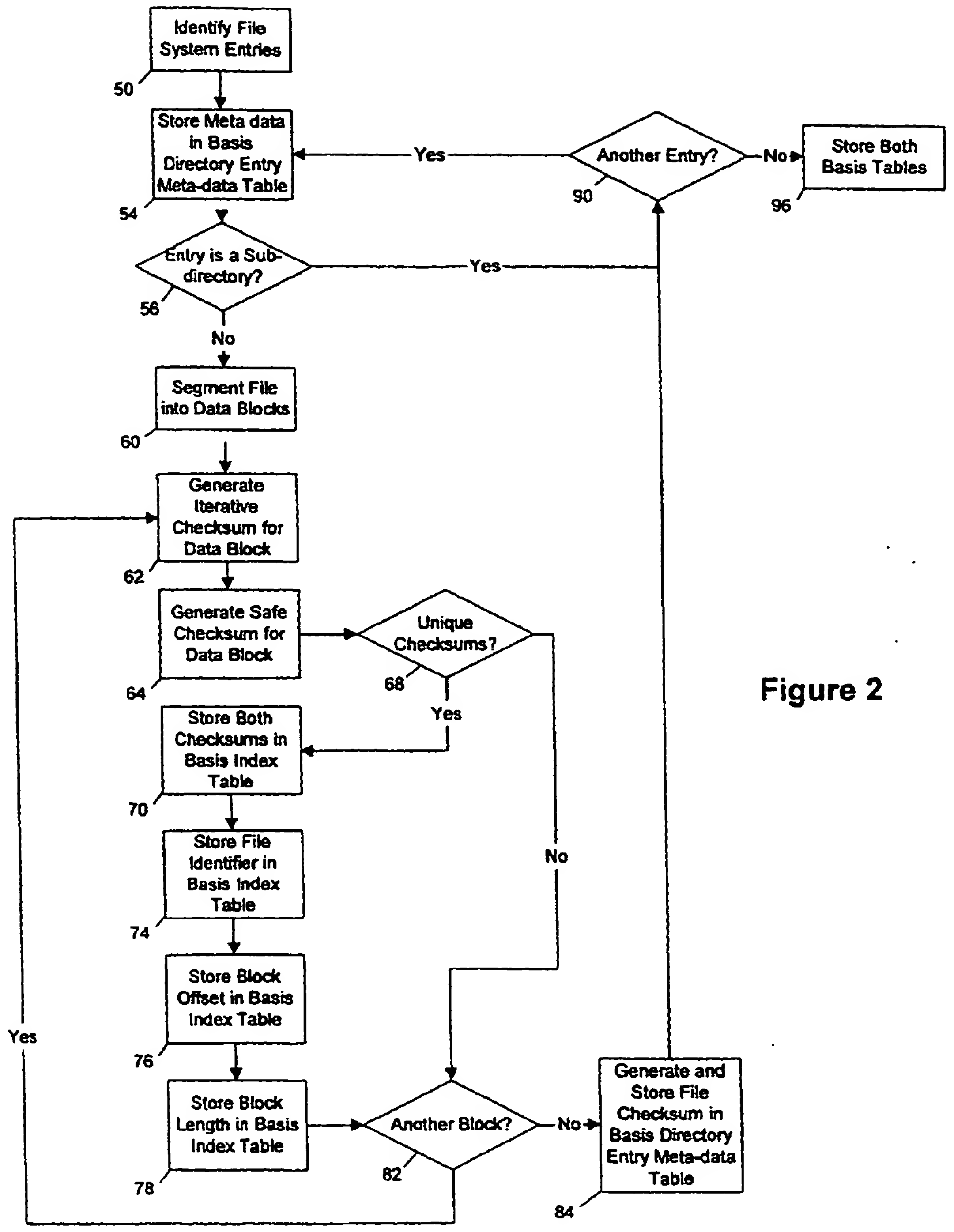
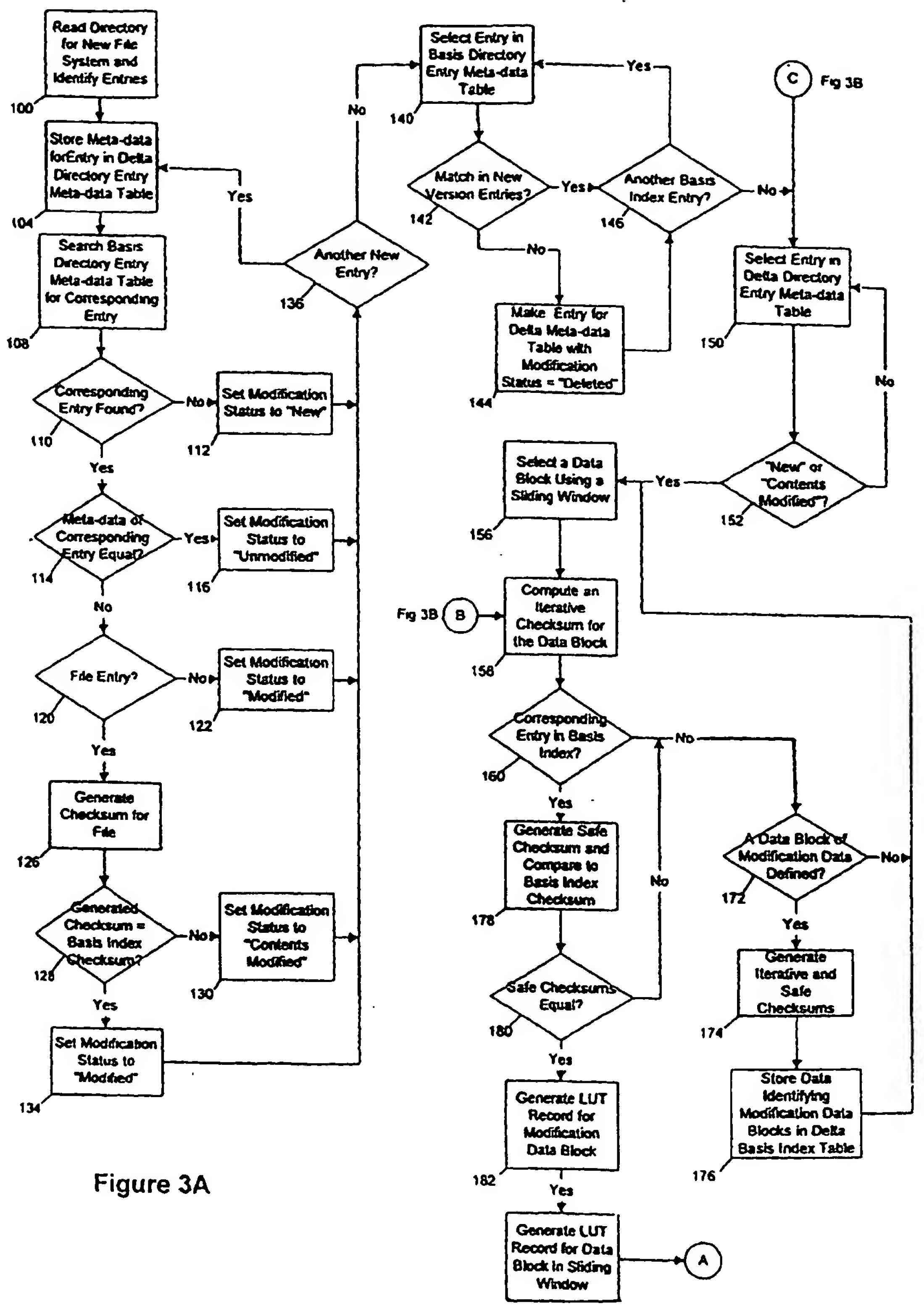

Identify File System Entries
50
Store Meta data in Basis Directory Entry Meta-data Table
54
Entry is a Sub-directory?
56
Yes
No
Segment File into Data Blocks
60
Generate Iterative Checksum for Data Block
62
Generate Safe Checksum for Data Block
64
Unique Checksums?
68
Yes
No
Store Both Checksums in Basis Index Table
70
Store File Identifier in Basis Index Table
74
Store Block Offset in Basis Index Table
76
Store Block Length in Basis Index Table
78
Another Block?
82
Yes
No
Generate and Store File Checksum in Basis Directory Entry Meta-data Table
84
Another Entry?
90
Yes
No
Store Both Basis Tables
96

Figure 2



Read Directory for New File System and Identify Entries
100
Store Meta-data forEntry in Delta Directory Entry Meta-data Table
104
Search Basis Directory Entry Meta-data Table for Corresponding Entry
108
Corresponding Entry Found?
110
No
Set Modification Status to "New"
112
Yes
Meta-data of Corresponding Entry Equal?
114
Yes
Set Modification Status to "Unmodified"
116
No
File Entry?
120
No
Set Modification Status to "Modified"
122
Yes
Generate Checksum for File
126
Generated Checksum = Basis Index Checksum?
128
No
Set Modification Status to "Contents Modified"
130
Yes
Set Modification Status to "Modified"
134
Another New Entry?
136
Yes
No
Select Entry in Basis Directory Entry Meta-data Table
140
Match in New Version Entries?
142
Yes
No
Make Entry for Delta Meta-data Table with Modification Status = "Deleted"
144
Another Basis Index Entry?
146
Yes
No
C
Fig 3B
Select Entry in Delta Directory Entry Meta-data Table
150
No
"New" or "Contents Modified"?
152
Yes
Select a Data Block Using a Sliding Window
156
Fig 3B
B
Compute an Iterative Checksum for the Data Block
158
Corresponding Entry in Basis Index?
160
No
Yes
Generate Safe Checksum and Compare to Basis Index Checksum
178
Safe Checksums Equal?
180
No
Yes
Generate LUT Record for Modification Data Block
182
Yes
Generate LUT Record for Data Block in Sliding Window
A
A Data Block of Modification Data Defined?
172
No
Yes
Generate Iterative and Safe Checksums
174
Store Data Identifying Modification Data Blocks in Delta Basis Index Table
176

Figure 3A

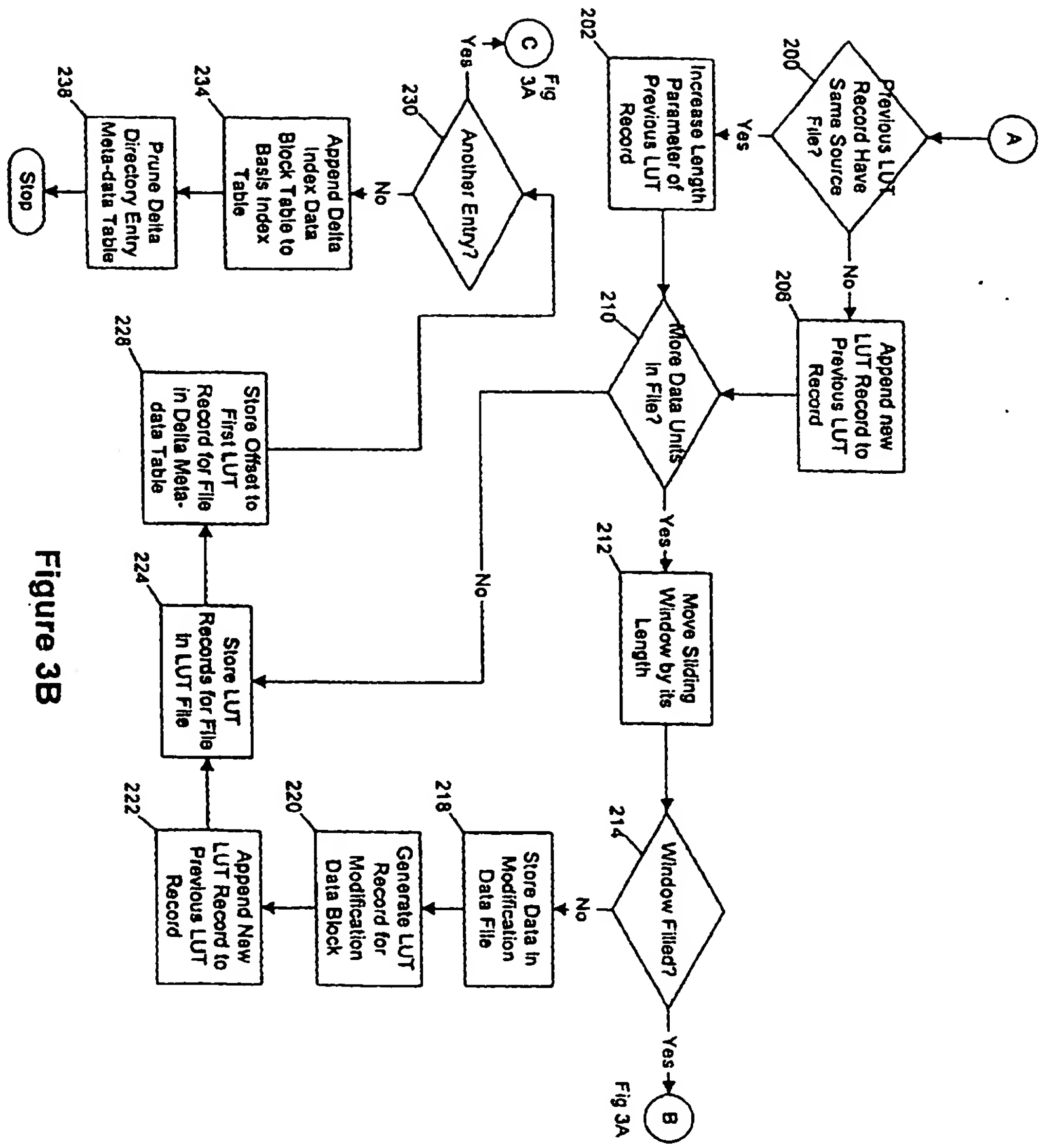

A
200
Previous LUT Record Have Same Source File?
Yes
No
202
Increase Length Parameter of Previous LUT Record
206
Append new LUT Record to Previous LUT Record
210
More Data Units in File?
Yes
No
212
Move Sliding Window by its Length
214
Window Filled?
Yes
B
Fig 3A
No
218
Store Data in Modification Data File
220
Generate LUT Record for Modification Data Block
222
Append New LUT Record to Previous LUT Record
224
Store LUT Records for File in LUT File
228
Store Offset to First LUT Record for File in Delta Meta-data Table
230
Another Entry?
Yes
C
Fig 3A
No
234
Append Delta Index Data Block Table to Basis Index Table
238
Prune Delta Directory Entry Meta-data Table
Stop

Figure 3B

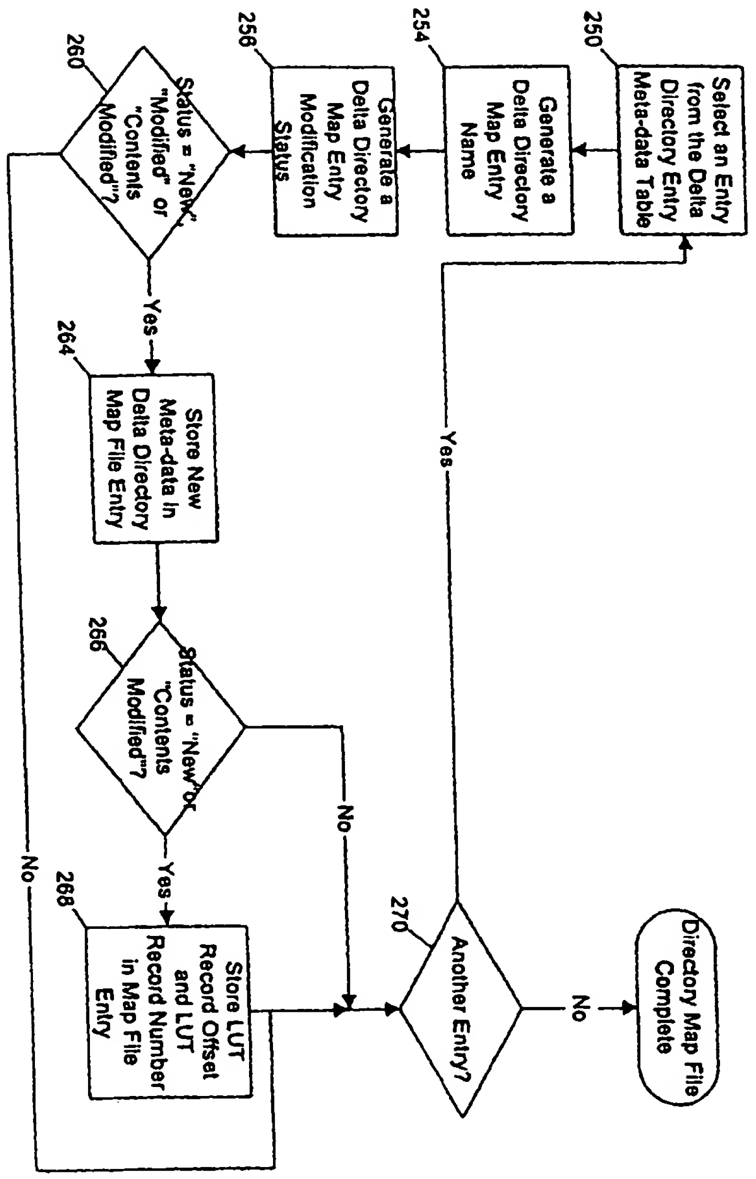

250
Select an Entry from the Delta Directory Entry Meta-data Table
254
Generate a Delta Directory Map Entry Name
256
Generate a Delta Directory Map Entry Modification Status
260
Status = "New", "Modified" or "Contents Modified"?
Yes
No
264
Store New Meta-data in Delta Directory Map File Entry
266
Status = "New" or "Contents Modified"?
Yes
No
268
Store LUT Record Offset and LUT Record Number in Map File Entry
270
Another Entry?
Yes
No
Directory Map File Complete

Figure 4

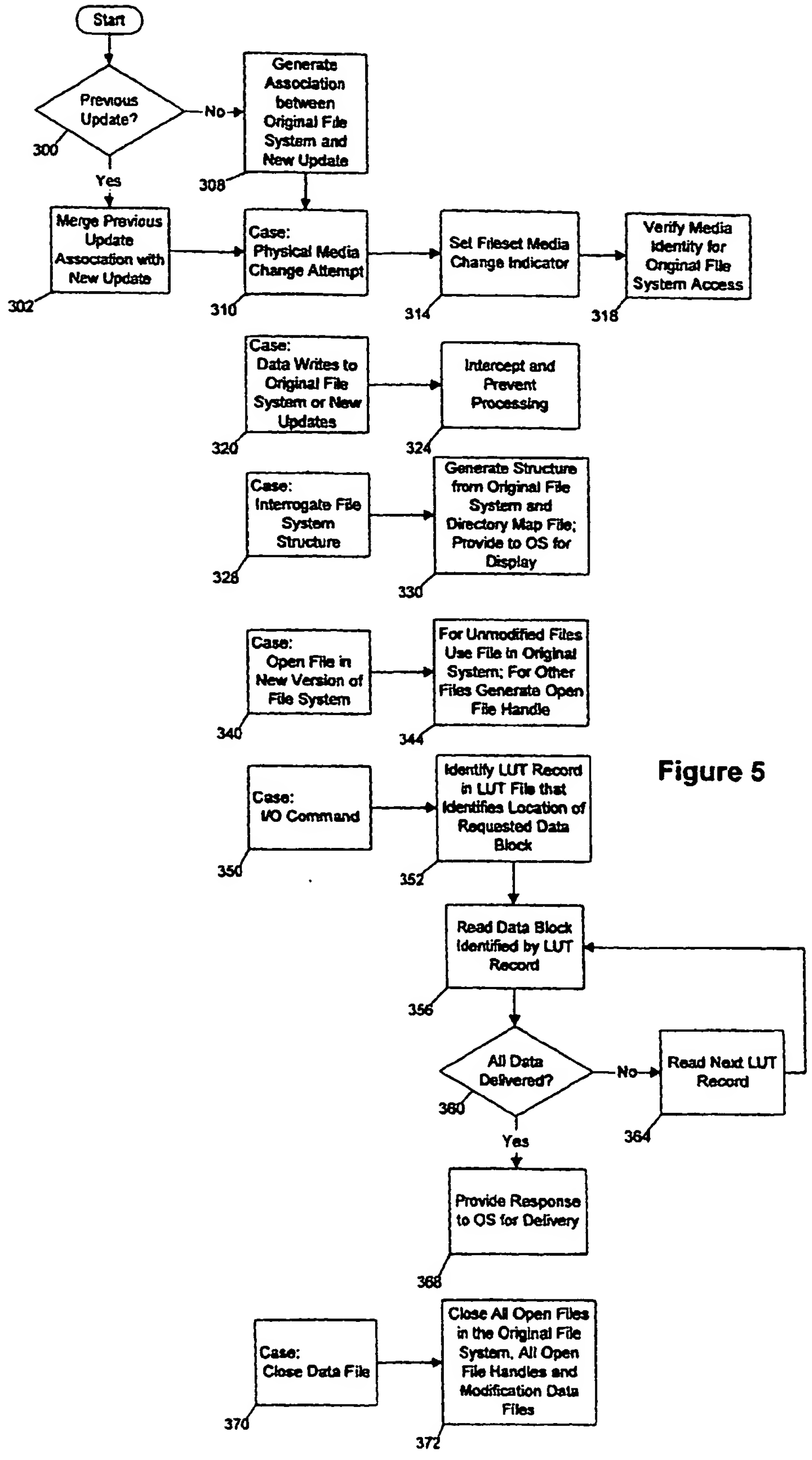

Start
Previous Update?
300
No
Yes
Generate Association between Original File System and New Update
308
Merge Previous Update Association with New Update
302
Case: Physical Media Change Attempt
310
Set Fileset Media Change Indicator
314
Verify Media Identity for Original File System Access
318
Case: Data Writes to Original File System or New Updates
320
Intercept and Prevent Processing
324
Case: Interrogate File System Structure
328
Generate Structure from Original File System and Directory Map File; Provide to OS for Display
330
Case: Open File in New Version of File System
340
For Unmodified Files Use File in Original System; For Other Files Generate Open File Handle
344
Case: I/O Command
350
Identify LUT Record in LUT File that Identifies Location of Requested Data Block
352
Read Data Block Identified by LUT Record
356
All Data Delivered?
360
No
Read Next LUT Record
364
Yes
Provide Response to OS for Delivery
368
Case: Close Data File
370
Close All Open Files in the Original File System, All Open File Handles and Modification Data Files
372

Figure 5